LOJ1108

The problem is about finding the shortest path in a graph where the cost of traveling between junctions is determined by a cubic function of their "busyness" difference. Specifically, moving from junction u to junction v has a cost of (busyness[v] - busyness[u])³. Because of the cubic nature, some edge costs may be negative when the destination is less busy than the source. These negative weights introduce the possibility of negative cycles, which complicates finding the true minimum cost.

The goal is to calculate the minimum total cost, or "earning" for the city authority, from junction 1 to a set of queried junctions. If a path is influenced by a negative cycle or results in a cost less than 3, the output should be ?. This means we must carefully detect and propagate the effect of negative cycles and handle unreachable nodes.

The approach can be summarized as follows:

1. **Input Handling**
    - Read the number of junctions and their busyness values.
    - Read the number of roads and store each as a directed edge with weight calculated as (busyness[destination] - busyness[source])³.
    - Read the queries for specific junctions.
2. **Initialization**
    - Set up a distance array with very large initial values.
    - Set the distance of junction 1 to 0.
    - Prepare an array to track nodes affected by negative cycles.
3. **Run Bellman-Ford**
    - Repeat n-1 times:
        - For each edge (u → v), update distance[v] if going through u offers a smaller cost.
4. **Detect Negative Cycles**
    - Perform an extra pass over all edges.
    - If any distance can still be improved, mark that node as part of a negative cycle.

- Propagate this influence using BFS or DFS to all nodes reachable from these cycle nodes.

5. **Respond to Queries**
    - For each query junction:
        - If the junction is unreachable, affected by a negative cycle, or the distance is below 3, output ?.
        - Otherwise, output the computed distance.
6. **Output**
    - Print the results with a "Set #" header for each test case, followed by the query answers on separate lines.

    Input:

    5 6 7 8 9 10

    6

    1 2

    2 3

    3 4

    1 5

    5 4

    4 5

    2

    4

    5

    Execution:

    Step 1: Read Input

    n = 5 busyness array: index 1=6, index 2=7, index 3=8, index 4=9, index

    5=10 r = 6 roads

    Compute edge weights:

1→2: (7-6)^3 = 1

2→3: (8-7)^3 = 1

3→4: (9-8)^3 = 1

1→5: (10-6)^3 = 64

5→4: (9-10)^3 = -1

4→5: (10-9)^3 = 1 q = 2

queries: 4 and 5

Step 2: Initialize Arrays

dist = [inf, 0, inf, inf, inf, inf] for nodes 1-5

inCycle = all false visited = all false

Step 3: Run Bellman-Ford (4 iterations)

Iteration 1:

Process edge 1→2: 0+1=1 < inf → update dist[2]=1

Process edge 2→3: 1+1=2 < inf → update dist[3]=2

Process edge 3→4: 2+1=3 < inf → update dist[4]=3

Process edge 1→5: 0+64=64 < inf → update dist[5]=64

Process edge 5→4: 64+(-1)=63 > 3 → no update Process

edge 4→5: 3+1=4 < 64 → update dist[5]=4

Iteration 2:

All edges checked, no improvements found

Iterations 3-4: No

changes

Current distances: dist[1]=0, dist[2]=1, dist[3]=2, dist[4]=3, dist[5]=4

Step 4: Detect Negative Cycles

Run Bellman-Ford again to check for cycles All

edges processed, no distance improvements

inCycle array remains all false

No BFS propagation needed since no cycles detected

Step 5: Process Queries

Query for node 4: dist[4]=3, not inf, not in cycle, ≥3 → output 3

Query for node 5: dist[5]=4, not inf, not in cycle, ≥3 → output 4

Output:

Set #1

3

4

Pseudocode: INPUT n

INPUT busyness values for junctions 1..n

INPUT r roads; for each road, store as edge with weight = (busyness[to] - busyness[from])³

INPUT q queries

distances[1..n] = very_large_value

distances[1] = 0

```
FOR i = 1 TO n-1:
   FOR each edge (from, to, weight):
      IF distances[from] + weight < distances[to]:
         distances[to] = distances[from] + weight
cycleFlag[1..n] = false
FOR i = 1 TO n-1:
   FOR each edge (from, to, weight):
      IF distances[from] + weight < distances[to]:
         cycleFlag[to] = true
         PROPAGATE_NEGATIVE_CYCLE(to, cycleFlag)
```

```
FOR each query node:
    IF distances[query] < 3 OR unreachable OR cycleFlag[query] == true:
        PRINT "?"
    ELSE:
        PRINT distances[query]
```

Here is the code for LOj 1108: https://github.com/pulokdas062/Graph-Algorithm/edit/main/Bellmanford/LOJ1108/LOJ1108.cpp